This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Currently Amended) In a runtime environment comprising a first program module, at least one second program module and a call stack, [[A]] a method of ~~using a first software module to~~ invok[[e]]ing a desired method associated with a desired second ~~software~~ program module, ~~the method~~ comprising:

in a third program module associating each of a plurality of stubs respectively with each of a plurality of methods associated with the second program module, wherein each stub comprises a code segment performing a unique non-standard calling convention into the second program module, wherein each stub includes at least a first instruction to push function parameters onto the call stack, a second instruction to call an authenticator module for authenticating that a stub has not been modified and a third instruction comprising embedded unique data for the stub, wherein the embedded data comprises a vtable entry descriptor for the desired method, corresponding to a vtable for the third module, wherein the vtable is covered and comprises a list of function pointers to functions associated with the second module arranged in a random order, the random order unique for each second module;

from the first ~~software~~ program module, issuing a first call to ~~a first method that invokes a functionality performed by the second software module~~a stub in the third program module associated with the desired method, whereupon after the first call, the call stack comprises at least a first parameter corresponding to a return address associated with the stub, a second parameter corresponding to a parameter depth (cArgs) and a third parameter corresponding to a return address of the first program module, the first, second and third parameters arranged in a top-down order;

~~using a third software module having one or more stubs that comprise code segments that are callable by the first software module as an intermediary, the one or more stubs for performing said first method, the one or more stubs being used to enter the second software module and identify said functionality, said call being made according to a first calling convention, said third software module using a second calling convention different from said first calling convention to invoke said functionality in the second software module, the~~

~~second calling convention comprising a non-standard calling convention that preserves a return address across more than one call boundary;~~

~~verifying that the call originated from a source that is permitted to invoke said functionality, the one or more stubs from the third software module comprising data required during said verification by the second software module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification;~~

wherein the third program module calls the second program module using a non-standard calling convention

~~performing said functionality that includes sensitive functions at the second software module, the sensitive functions comprising verifying and authenticating the call from the first software module; and~~

~~returning from said second software module to said first software module that issued the call and bypassing said third software module and the one or more stubs~~.

2. (Currently Amended) The method of claim 1, wherein ~~said first method is performed by the second software module, said first method being exposed to the first software module, said first method performing said functionality~~ upon returning to the authenticator module from the jump to the vtable uncovering code, the address associated with the desired method on the program stack automatically causes calling of the desired method associated with the third module and whereupon completion of the desired method, the second return address on the program stack automatically causes return to the cleanup function, whereupon completion of the portion of the authenticator corresponding to the cleanup function, the first address associated with the first program module on the program stack causes return to the first program module.

3-4. (Canceled)

5. (Currently Amended) The method of claim 1, wherein ~~said second calling convention causes a program stack to be modified in order to cause a next occurring return~~

~~instruction to cause a return to the location in which a return would have occurred if a return had been executed following a call to said third software module and prior to a call to said second software module~~the desired method performs static authentication of the first program module.

6. (Currently Amended) The method of claim 5, wherein ~~said first calling convention comprises placing a first return address on a stack, said first return address representing a location at which execution is to resume after a function call is completed, and wherein said second calling convention comprises placing a second return address on a stack and placing data at the location represented by said second return address, and wherein the method further comprises:~~

~~using said second return address to find said data, said data being used for at least one of:~~

~~performing said verifying act; and~~

~~identifying a location to execute in order to perform said functionality~~~~module~~the desired method performs dynamic authentication of the first program module.

7. (Currently Amended) The method of claim [[1]]7, wherein said ~~verifying act~~ authentication comprises:

examining [[a]] the call stack to identify a return address, and determining that the return address is part of a program module that is permitted, according to a standard or rule, to invoke ~~said~~ functionality associated with the second program module.

8. (Currently Amended) The method of claim 7, further comprising:

determining that said return address is from a location or range of locations within said first ~~software~~ program module from which invocation of said functionality is permitted to originate.

9. (Currently Amended) The method of claim 1, further comprising:

verifying that said first ~~software~~ program module, or a portion thereof, has not been modified relative to a previously-known state.

10. (Currently Amended) The method of claim 1, wherein said first ~~software~~ program module is, or is part of, an application program.

11. (Currently Amended) The method of claim 1, wherein said second ~~software~~ program module comprises a dynamic-link library.

12. (Currently Amended) A method of verifying a context in which a first program module has been called, the method comprising:

examining a call stack of a process in which said first program module executes to identify a return address in which control of the process will return upon completion of a call to said first program module;

determining that said return address is located within a second program module that is permitted to call said first program module, said determining comprising checking a datum that represents a calling code used by the second program module, the datum being derived from a portion or the entirety of the second program module, the first program module being called by the second program module via a third program module having one or more stubs with code segments that are callable by the second program module as an intermediary, the one or more stubs comprising data required during a verification by the first software module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification wherein each stub comprises a code segment performing a unique non-standard calling convention into the second program module, wherein each stub includes at least a first instruction to push function parameters onto the call stack, a second instruction to call an authenticator module for authenticating that a stub has not been modified and a third instruction comprising embedded unique data for the stub, wherein the embedded data comprises a vtable entry descriptor for the desired method, corresponding to a vtable for the third module, wherein the vtable is covered and comprises a

list of function pointers to functions associated with the second module arranged in a random order, the random order unique for each second module; and

based on the result of said determining act, permitting execution of said first program module to proceed and returning to said second software module which issued the call and bypassing said third software module and the one or more stubs,

wherein said first program module comprises cryptographic functionality that stores and obscures a decryption key and that uses said decryption key to decrypt content.

13. (Previously Presented) The method of claim 12, further comprising:

determining that said second program module, or a portion thereof, has not been modified relative to a previously-known state of said second program module, wherein said act of permitting execution of said first program module to proceed is further based on the determination as to whether said second program module has been modified.

14. (Original) The method of claim 12, wherein said first program module includes logic that resists misuse and/or tampering with said first program module, and wherein said determining act is performed by said first program module.

15. (Cancelled)

16. (Original) The method of claim 12, wherein said first program module is called by a third program module, said third program module having a callable method exposed to said second program module, said callable method causing said first program module to be invoked and passing said return address to said first program module at the time that said first program module is invoked.

17. (Original) The method of claim 16, wherein said first program module adjusts the content of said call stack to reflect that said first program module will return to said return address upon completion of execution of said call to said first program module, said call stack reflecting, in the absence of the adjustment, that said first program module will return to an address other than said return address.

18. (Currently Amended) A program module stored in a computer-readable storage medium comprising:

a function that is performable on behalf of a calling entity; and

logic that verifies an identity of the calling entity as a condition for performing said function, said logic consulting a call stack in order to identify said calling entity and determining said identity based on a return address on said call stack, said return address representing a location of an instruction to be executed when the program module completes execution, said logic checking a datum that represents a calling code used by the calling entity, the datum being derived from a portion or the entirety of the calling entity;

wherein said function is not exposed to said calling entity, and wherein said function is exposed to an intermediate entity that is callable by said calling entity, said intermediate entity calling upon the program module to perform said function on behalf of said calling entity, said intermediate entity comprising one or more stubs that comprise data required by the logic to verify the identity of the calling entity, the data being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify the function, wherein each stub comprises a code segment performing a unique non-standard calling convention into the second program module, wherein each stub includes at least a first instruction to push function parameters onto the call stack, a second instruction to call an authenticator module for authenticating that a stub has not been modified and a third instruction comprising embedded unique data for the stub, wherein the embedded data comprises a vtable entry descriptor for the desired method, corresponding to a vtable for the third module, wherein the vtable is covered and comprises a list of function pointers to functions associated with the second module arranged in a random order, the random order unique for each second module;

wherein the program module upon completing said function bypasses the intermediate entity and returns to the calling entity's return address.

19-20. (Canceled)

21. (Previously Presented) The program module of claim 18, wherein said calling entity calls said intermediate entity using a first calling convention, and wherein said intermediate entity calls the program module using a second calling convention different from said first calling convention.

22. (Previously Presented) The program module of claim 21, wherein said calling entity calls said intermediate entity by calling a function in said intermediate entity and placing a first return address on said call stack, and wherein said intermediate entity calls the program module by calling or jumping to a location in said program module with one or more parameters including said first return address, wherein the program module verifies that said first return address is located within a calling entity that is allowed to call the program module, and wherein the program module adjusts said call stack so that a return address to be followed upon a next return instruction is equal to said first return address.

23. (Currently Amended) The program module of claim 21, wherein said first calling convention comprises placing a return address on said call stack, and wherein said second calling convention comprises placing a second return address on said call stack and placing data at the location represented by said second return address, said second return address being used to find said data, said data being used to perform at least one of: verification of the identity of the calling entity; and identification of a location at which said function is located.

24. (Currently Amended) A computer-readable storage medium storing computer-executable instructions to perform a method that facilitates verification of a call stack, the method comprising:

receiving a first call or first jump from a first entity, there being a call stack which, at the time of said first call or first jump, has a state in which a return address to be executed upon a next return instruction is equal to a first value; and

issuing a second call or a second jump to a second entity, the second call or second jump being parameterized by one or more values including said first value, said second entity having access to a second value and using said second value to verify which return address was applicable at the time that said first call or said first jump was made, said second entity

adjusting said call stack to set the return address equal to the first value,

wherein said second entity returns directly to said first entity without returning to an intermediate entity that received said first call or said first jump by causing a jump into vtable uncovering code associated with a vtable entry.

25. (Canceled)

26. (New). The method of claim 1, ,whereupon executing the second instruction in the stub, an authenticator module is called, the authenticator:

verifying that the associated stub has not been modified utilizing the return address of the associated stub;

if verification is not successful, not calling the second program module and terminating execution;

if verification is successful:

inserting a first return address of the first module on the call stock;

replacing the stub return address on the call stack with an address associated with the desired method;

replacing the second parameter (cArgs) on the call stack with a second return address associated with the authenticator, the second return address corresponding to a portion of the authenticator for performing a cleanup function;

causing a jump into vtable uncovering code associated with the vtable entry, causing execution of the desired method, automatically bypassing the authenticator module upon return and automatically calling the cleanup function, wherein the desired method authenticates the first program module using the return address of the first program module having been preserved on the call stack.